



Bahasa C++: Konsep Kelas (bagian I)

IF2210 – Semester II 2022/2023

Sumber: Diktat Bahasa C++ oleh Hans Dulimarta

Latar Belakang C++

- › Diciptakan oleh Bjarne Stroustrup di AT&T Bell Laboratories pada awal 1980an
- › Pada mulanya dikenal sebagai "C with Classes" (nama C++ digunakan sejak 1983, setelah diusulkan oleh Rick Mascitti).
- › 1985: disebarluaskan oleh AT&T, perangkat lunak cfront (C++ translator).
- › Didasarkan pada bahasa C, Simula67, Algol68, Ada.
 - › Simula67: konsep kelas.
 - › Algol68: konsep operator overloading dan kemungkinan penempatan deklarasi di manapun
 - › Ada: konsep template dan exception

Perbandingan C++ dengan C (1)

- › *Typecasting* dalam C++ dapat dipandang sebagai fungsi.
 - › e.g. `(int) x` vs. `int(x)`
- › *Function name overloading*: fungsi dengan nama yang sama namun dengan *signature* yang berbeda.
- › Nilai default pada parameter formal.
 - › e.g. `int f(int a, int b=2) { ... }` jika dipanggil dengan `f(5)` maka `a` bernilai 5 dan `b` bernilai 2.
- › *Template function, operator function, inline function.*
- › *Reference variable*, dan *call by reference* (berbeda dengan *address of*).

Perbandingan C++ dengan C (2)

- › Operator baru seperti *global scope* (unary `::`), *class scope* (binary `::`), `new`, `delete`, member pointer selectors (`->*`, `.*`) dan kata kunci baru seperti: `class`, `private`, `operator`, dsb.
- › Nama kelas atau enumerasi (*tag name*) adalah nama tipe (baru)
- › *Anonymous union*

Reference (1)

- › Reference variable: nama alias terhadap variabel tsb.
 - › Jika sudah digunakan untuk mengacu suatu objek/variabel, reference tidak dapat direset untuk mengacu objek/variabel lain
 - › Setiap pendefinisian reference variable harus selalu diinisialisasi dengan variabel lain

```
int x = 5;  
int &xr = x; // xr mengacu pada x  
xr++;       // xr merupakan alias dari x
```

Reference (2)

- › Penggunaan *reference* lain: untuk *call-by-reference* dan *return value* dari sebuah fungsi
- › *Reference* **berbeda** dengan pointer
- › Dalam C++ simbol & digunakan dengan 2 makna: *address-of* dan *reference*

```
int *py;  
int &yr; // error (tidak diinisialisasi)
```

```
int y;  
py = &y; // py akan berisi alamat dari y
```

Kompatibilitas antara C++ dan C

- › Program C yang dikompilasi oleh C++ tidak dapat menggunakan kata kunci dari C++ sebagai nama identifier
- › Setiap fungsi harus dideklarasikan (harus memiliki *prototype*)
- › Fungsi yang bukan bertipe `void`, harus memiliki instruksi `return`
- › Penanganan inisialisasi array karakter:

```
char ch[3] = "C++"; /* C: OK, C++: error */  
char ch[] = "C++"; /* OK untuk C dan C++ */
```

Class

- › Untuk menciptakan tipe data baru.
- › Berisi operasi-operasi dan data yang akan dimiliki objek-objek yang berasal dari kelas tsb.
 - › **Operasi** dalam bentuk **method**/function member biasanya public.
 - › **Data** dalam bentuk **atribut**/data member sebaiknya private. (Secara konsep OOP atribut seharusnya private, namun fitur bahasa C++ mengizinkan atribut public).
- › Peran: perancang kelas dan pengguna kelas.
 - › **Perancang**: menentukan operasi yang disajikan pada pengguna kelas serta representasi internal objek.
 - › **Pengguna**: memanfaatkan operasi tersebut untuk memanipulasi objek.

Class vs Struct (1)

Struct	Class
Memiliki ≥ 1 <i>field</i> , masing-masing berupa data	Memiliki ≥ 1 <i>member</i> , masing-masing berupa data atau fungsi
Setiap field dapat diacu secara bebas dari luar	Pengaksesan member dari luar dapat dikendalikan (kata kunci <i>private</i> , <i>public</i> , dan <i>protected</i>)

Pada class, ada dua jenis member:

- **Function member**, kumpulan operasi (*service/method*) yang dapat diterapkan terhadap objek, seringkali disebut juga sebagai *class interface*
- **Data member**, yang merupakan representasi internal dari kelas (*atribut*)

Class vs Struct (2)

Struct	Class
<pre>typedef struct { int x; int y; } Point; void moveTo (Point&) { // ... }</pre>	<pre>class Point { int x; int y; void moveTo () { // ... } };</pre>

Pada class:

- Pengaturan akses terhadap anggota kelas: `private`, `public`, `protected`
- Perhatikan perubahan parameter aktual pada prosedur `MoveTo`

Class

- › Pengaturan akses terhadap anggota kelas: `private`, `public`, `protected`

Wilayah Deklarasi	Makna
<code>public</code>	Dapat diakses oleh fungsi di luar kelas dengan menggunakan operator selektor (<code>.</code> atau <code>-></code>) “Fungsi luar”: fungsi yang bukan anggota kelas tersebut
<code>private</code>	hanya dapat diakses oleh fungsi kelas tersebut
<code>protected</code>	seperti <code>private</code> , namun dapat diakses oleh kelas turunan

- › Pendefinisian member function dapat dilakukan dengan dua cara:
 - › Di dalam class body, otomatis menjadi inline function
 - › Di luar class body, nama fungsi harus didahului oleh class scope. Di dalam kelas hanya dituliskan prototipe fungsi

Deklarasi Kelas Stack + definisi method

```
class Stack {  
    public:  
        // methods  
        void pop(int& ); // deklarasi (prototype)  
        void push(int); // deklarasi (prototype)  
        /*--- pendefinisian method di dalam class body ---*/  
        int is_empty() {  
            return topStack == 0;  
        }  
    private:  
        // attributes  
        int topStack; /* posisi yang akan diisi berikutnya */  
        int *data;  
}; // PERHATIKAN TITIK KOMA !!!
```

Pendefinisian method di luar Class

```
/* pendefinisian Pop dan Push di
luar class body */
void Stack::pop(int& item) {
    if (is_empty()) {
        // error message
    } else {
        topStack--;
        item = data [topStack];
    }
} // TIDAK PERLU TITIK KOMA !!!
```

```
void Stack::push (int item) {
    if (/* penuh */) {
        // error message
    } else {
        data [topStack] = item;
        topStack++;
    }
}
```

Pointer Implisit `this`

- › Setiap *method* memiliki pointer ke objek `this`, yang secara implisit pointer ini dideklarasikan sebagai (untuk kelas X):

`X* this`

- › Pengaksesan anggota kelas (method/atribut) dapat dituliskan dengan menyertakan `this->`

```
void Stack::push (int item) {  
    //...  
    this->data [this->topStack] = item;  
    this->topStack++;  
    //...  
}
```

- › Manfaat: Setiap objek memiliki atribut terpisah, namun method bersama, `this` diperlukan untuk mengakses atribut.

Objek dari Kelas

- › Contoh deklarasi objek:

```
Stack myStack;  
Stack oprStack[10];  
Stack *pts = new Stack;  
Stack ns = myStack; // definition & initialization
```

- › Pengaksesan anggota public:

```
int x;  
myStack.push(99);  
oprStack[2].pop(x);  
pts->push(x);  
if (myStack.is_empty()) {  
    printf ("Stack masih kosong");  
}
```

Penulisan Kode Kelas

- › Kode untuk kelas terdiri dari
 1. Interface / specification: **deklarasi** kelas. Dituliskan ke dalam file **X.h**.
 2. Implementation / body : **definisi** dari method-method dari kelas tersebut. Dituliskan ke dalam file **X.cc**, **X.cpp**, **X.cxx** atau **X.c**.
- › Untuk mencegah penyertaan header lebih dari satu kali, deklarasi kelas dituliskan di antara `#ifndef XXXX_H` dan `#endif`

Contoh Header File

```
// Nama file: Stack.h
// Deskripsi: interface dari kelas Stack
#ifndef STACK_H
#define STACK_H
class Stack {
    //
    // daftar signature method ("protocol") kelas Stack
    //
};
#endif
```

PERHATIAN: Di dalam header file, jangan menuliskan **definisi** objek/variabel karena akan mengakibatkan kesalahan “*multiply defined name*” pada saat linking dilakukan.

Contoh Implementation File

```
// Nama file: Stack.cc
// Deskripsi: implementasi/body dari kelas Stack

#include "Stack.h"

Stack::Stack() {
    // ... ctor
}

Stack::~~Stack() {
    // ... dtor
}

// dst...
```

Pemanfaatan Kelas

› Perancang kelas:

1. Kompilasi file implementasi (*.cpp) menjadi kode objek (*.o)
2. Berikan file header (*.h) dan file kode objek (*.o) kepada pemakai kelas (mungkin dalam bentuk *library*)

› Pemakai kelas:

1. Sertakan file header di dalam program (main.cc)
2. Kompilasi program C++ menjadi kode objek (main.o)
3. Link main.o dengan kode objek yang diberikan perancang kelas

Next Topic

- › 4 sekawan:
 - › ctor, cctor, dtor, operator=

Tugas Baca #2

- › “Konsep tambahan paradigma objek (1)” (3 halaman)
- › Buat summary, 1-2 kalimat untuk setiap *heading*.
- › Kumpulkan di Edunex.